Streaming Periodicity with Mismatches*†

Funda Ergün[1], Elena Grigorescu[2], Erfan Sadeqi Azer[3], and Samson Zhou[4]

1 **School of Informatics and Computing, Indiana University, Bloomington, IN, USA**
`fergun@indiana.edu`
2 **Department of Computer Science, Purdue University, West Lafayette, IN, USA**
`elena-g@purdue.edu`
3 **School of Informatics and Computing, Indiana University, Bloomington, IN, USA**
`esadeqia@indiana.edu`
4 **Department of Computer Science, Purdue University, West Lafayette, IN, USA**
`samsonzhou@gmail.com`

Abstract

We study the problem of finding all k-periods of a length-n string S, presented as a data stream. S is said to have k-period p if its prefix of length $n-p$ differs from its suffix of length $n-p$ in at most k locations.

We give a one-pass streaming algorithm that computes the k-periods of a string S using $\text{poly}(k, \log n)$ bits of space, for k-periods of length at most $\frac{n}{2}$. We also present a two-pass streaming algorithm that computes k-periods of S using $\text{poly}(k, \log n)$ bits of space, regardless of period length. We complement these results with comparable lower bounds.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases String algorithms, Streaming algorithms, Pattern matching, Randomized algorithms, Sublinear algorithms

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2017.42

1 Introduction

In this paper we are interested in finding (possibly imperfect) periodic trends in sequences given as streams. Informally, a sequence is said to be *periodic* if it consists of repetitions of a block of characters; e.g., *abcabcabc* consists of repetitions of *abc*, of length 3, and thus has period 3. The study of periodic patterns in sequences is valuable in fields such as string algorithms, time series data mining, and computational biology. The question of finding the smallest period of a string is a fundamental building block for many string algorithms, especially in pattern matching, such as the classic Knuth-Morris-Pratt [21] algorithm. The general technique for many pattern matching algorithms is to find the periods of prefixes of the pattern in a preprocessing stage, then use them as a guide for ruling out locations where the pattern cannot occur, thus improving efficiency.

* A full version of the paper is available at `http://homes.soic.indiana.edu/fergun/PUBLICATIONS/mismatchperiodicity.pdf`.

† Funda Ergün's research is supported by NSF CCF-1619081; Elena Grigorescu's and Samson Zhou's research is supported by NSF CCF-1649515.

While finding exact periods is fundamental to pattern matching, in real life, it is unrealistic to expect data to be perfectly periodic. In this paper, we assume that even when there is a fixed period, data might subtly change over time. In particular, we might see *mismatches,* defined as locations in the sequence where a block is not the same as the previous block. For instance, while *abababababab* is perfectly periodic, *abababacacac* contains one mismatch where *ab* becomes (and stays) *ac*. This model captures periodic events that undergo permanent modifications over time (e.g., statistics that remain generally cyclic but experience infrequent permanent changes or errors). We consider our problem in the *streaming* setting, where the input is received in a sequential manner, and is processed using sublinear space.

Our problem generalizes exact periodicity studied in [12], where the authors give a one-pass, $\mathcal{O}\left(\log^2 n\right)$-space algorithm for finding the smallest *exact* period of stream S of length n, when the period is at most $n/2$, as well as a linear space lower bound when the period is longer than $n/2$. They use two standard and equivalent definitions of periodicity: S has period p if it is of the form $B^\ell B'$ where B is a block of length p that appears $\ell \geq 1$ times in a row, and B' is a prefix of B. For instance, *abcabcabcab* has period 3 where $B = abc$, and $B' = ab$. Equivalently, the length $n-p$ prefix of S is identical to its length $n-p$ suffix. These definitions imply that at most k of the repeating blocks differ from the preceding ones. According to this definition, for instance, *abcabdabdae* is 2-periodic with period 3, with the mismatches occurring at positions 6 and 11.

In order to allow mismatches in S while looking for periodicity in small space, we utilize the fingerprint data structure introduced for pattern matching with mismatches by [25, 7]. Ideally, one would hope to combine results from [12] and [7] to readily obtain an algorithm for detecting k-periodicity. Unfortunately, reasonably direct combinations of these techniques do not seem to work. This is due to the fact that, in the presence of mismatches, the essential structural properties of periods break down. For instance, in the exact setting, if S has periods p and q, it must also have period r, where r is any positive multiple of p or q. It must also have period $d = gcd(p, q)$. These are not necessarily true when there are mismatches; as an example consider the following.

▶ **Example 1.** $S = aaaaba$ has only one mismatch where $S[i] \neq S[i+2]$ (over all non range-violating values of i); likewise where $S[i] \neq S[i+3]$, thus S is 1-periodic with periods 2 and 3. S is *not* 1-periodic with period $1 = \mathsf{gcd}\,(2,3)$ as it has *two* mismatches where $S[i] \neq S[i+1]$.

In the exact setting the smallest period t determines the entire structure of S as all other periods must be multiples of t. This property does not necessarily hold when we allow mismatches, thus the smallest period does not carry as much information as in the exact case. Similarly, overlaps of a pattern with itself in S exhibits a much less well-defined periodic structure in the presence of mismatches. This makes it much harder to achieve the fundamental space reduction achievable in exact periodicity computation, where this kind of structure is crucially exploited.

1.1 Our Results

Given the structural challenges introduced by the presence of mismatches, we first focus on understanding the unique structural properties of k-periods and the relationship between the period p, and the number of mismatches k (See Theorem 9). This understanding gives us tools for “compressing” our data into sublinear space. We proceed to present the following on a given stream S of length n:

1. a two-pass streaming algorithm that computes all k-periods of S using $\mathcal{O}\left(k^4 \log^9 n\right)$ space, *regardless of period length* (see Section 4)
2. a one-pass streaming algorithm that computes all k-periods of length at most $n/2$ of S using $\mathcal{O}\left(k^4 \log^9 n\right)$ space (see Section 5)
3. a lower bound that any one-pass streaming algorithm that computes all k-periods of S requires $\Omega(n)$ space (see Section 6)
4. a lower bound that for $k = o(\sqrt{n})$ with $k > 2$, any one-pass streaming algorithm that computes all k-periods of S with probability at least $1 - 1/n$ requires $\Omega(k \log n)$ space, even under the promise that the k-periods are of length at most $n/2$. (see Section 6)

Given the above results, it is trivial to modify the algorithms to return, rather than all k-periods, the smallest, largest, or any particular k-period of S.

1.2 Related Work

Our work extends two natural directions in sublinear algorithms for strings: on one hand the study of the repetitive structure of long strings, and on the other hand the notion of approximate matching of patterns, in which the algorithm can detect a pattern even when some of it got corrupted.

In the first line of work, Ergün *et al.* [12] initiate the study of streaming algorithms for detecting the period of a string, using $poly(\log n)$ bits of space. Indyk *et al.* [19] also studied mining periodic patterns in streams, [10] studied periodicity in time-series databases and online data, and Crouch and McGregor [9] study periodicity via linear sketches. [13] and [23] studied the problem of distinguishing periodic strings from aperiodic ones in the property testing model of sublinear-time computation. Furthermore, [1] studied approximate periodicity in RAM model under the Hamming and swap distance metrics.

The pattern matching literature is a vast area (see [3] for a survey) with many variants. Following the pattern matching streaming algorithm of Porat and Porat [25], Clifford *et al.* [7] recently show improved streaming algorithms for the k-mismatch problem, as well as offline and online variants. We adapt the use of sketches from [7] though there are some other works with different sketches for strings ([2], [5], [27] and [26]). [8] also showed several lower bounds for online pattern matching problem.

This line of work is also related to the detection of other natural patterns in strings, such as palindromes or near palindromes. Ergün *et al.* [4] initiate the study of this problem and give sublinear-space algorithms, while [16] show lower bounds. In recent work, [18] extend this problem to finding near-palindromes (i.e., palindromes with possibly a few corrupted entries).

Many ideas used in these sublinear algorithms stem from related work in the classical offline model. The well-known KMP algorithm [22] initially used periodic structures to search for patterns within a text. Galil *et al.*[14] later improved the space performance of this pattern matching algorithm. Recently, [15] also used the properties of periodic strings for pattern matching when the strings are compressed. These interesting properties have allowed several algorithms to satisfy some non-trivial requirements of respective models (see [17], [6] for example).

2 Preliminaries

We assume our input is a stream $S[1, \ldots, n]$ of length $|S| = n$ over some alphabet Σ. The i^{th} character of S is denoted $S[i]$, and the substring between locations i and j (inclusive) $S[i, j]$. Two strings $S, T \in \Sigma^n$ are said to have a *mismatch* at index i if $S[i] \neq T[i]$, and their Hamming distance is the number of such mismatches, denoted $\mathsf{HAM}(S,T) = \Big|\{i \mid S[i] \neq T[i]\}\Big|$. We denote the concatenation of S and T by $S \circ T$.

S is said to have *period* p if $S[x] = S[x+p]$ for all $1 \leq x \leq n-p$; more succinctly, if $S[1, n-p] = S[p+1, n]$. In general, we say S has k-period p (i.e., S has period p with k mismatches) if $S[x] = S[x+p]$ for all but at most k valid indices x. Equivalently, S has k-period p if and only if $\mathsf{HAM}(S[1, n-p], S[p+1, n]) \leq k$.

▶ **Observation 2.** *If p is a k-period of S, then at most k of the sequence of substrings $S[1,p], S[p+1,2p], S[2p+1,3p], \ldots$ can differ from the previous substring in the sequence.*

When obvious from the context, given k-period p, we denote as a *mismatch* a position i for which $S[i] \neq S[i+p]$.

▶ **Example 3.** The string $S = aaaaaabbccd$ has 3-period equal to 1, since $S[i] = S[i+1]$ for all valid locations i except mismatches at $i = 6, 8, 10$. On the other hand, $S = abcabcadcabc$ has 2-period equal to 3 since $S[i] = S[i+3]$ for all valid i except mismatches $i = 5, 8$.

The following observation notes that the number of mismatches between two strings is an upper bound on the number of mismatches between their prefixes of equal length.

▶ **Observation 4.** *If p is a k-period of S, then for any $x \leq n-p$, the number of mismatches between $S[1,x]$ and $S[p+1, p+x]$ is at most k.*

Given two integers x and y, we denote their greatest common divisor by $\mathsf{gcd}(x,y)$.

We repeatedly use data structures and subroutines that use Karp-Rabin fingerprints. For more about the properties of Karp-Rabin fingerprints see [20], but for our purposes, the following suffice:

▶ **Theorem 5** ([7])**.** *Given two strings S and T of length n, there exists a data structure that uses $\mathcal{O}\left(k \log^6 n\right)$ bits of space, and outputs whether $\mathsf{HAM}(S,T) > k$ or $\mathsf{HAM}(S,T) \leq k$, along with the set of locations of the mismatches in the latter case.*

From here, we use the term *fingerprint* to refer to this data structure.

2.1 The k-Mismatch Algorithm

For our string-matching tasks, we utilize an algorithm from [7], whose parameters are given in Theorem 6. For us, string matching is a tool rather than a goal; as a result, we require additional properties from the algorithm that are not obvious at first glance. In Corollary 7 we consider these properties. Throughout our algorithms and proofs, we frequently refer to this algorithm as the *k-Mismatch Algorithm.*

▶ **Theorem 6** ([7])**.** *Given a pattern P of length ℓ, a text T of length n and some mismatch threshold k, there exists an algorithm that, with probability $1 - \frac{1}{n^2}$, outputs all indices i such that $\mathsf{HAM}(T[i, i+\ell-1], P) \leq k$ using $\mathcal{O}\left(k^2 \log^8 n\right)$ bits of space.*

Whereas the pattern in the k-Mismatch Algorithm is given in advance and can be preprocessed before the text, in our case the pattern is a prefix of the text, and the algorithm must return any matches of this pattern, starting possibly at location 2, well within the original occurrence of the pattern itself. (Consider text 'abcdabcdabcdabcd' and the pattern 'abcdabcd,' the first six characters of the text. The first match starts at location 4, but the algorithm does not finish reading the full pattern until it has read location 6.) To eliminate a potential problem due to this requirement, we make modifications so that the algorithm can search for all matches in S of a prefix of S.

▶ **Corollary 7.** *Given a string S and an index x, there exists an algorithm which, with probability $1 - \frac{1}{n^2}$, outputs all indices i where* $\mathsf{HAM}\left(S[1, x], S[i + 1, i + x]\right) \le k$ *using* $\mathcal{O}\left(k^2 \log^8 n\right)$ *bits of space.*

Proof. We claim that the algorithm of Theorem 6 can be arranged and modified to output all such indices i. We need to input $S[1, x]$ as the pattern and $S[2, n]$ as the text for this algorithm.

Thus, it suffices to argue that the data structure for the pattern is built in an online fashion. That is, after reading each symbol of the pattern, the data structure corresponding to the prefix of the pattern that has already been read is updated and ready to use. Moreover, the process of building the data structure for the text should not depend on the pattern. The only dependency between these two processes can be that they need to use the same randomness. Therefore, the algorithm only needs to decide the randomness before starting to process the input and share it between processes.

The algorithm of Theorem 6 has a few components, explained in the proof of Theorem 1.2 in [7]. Here, we go through these components and explain how they satisfy the conditions we mentioned.

The main data structure for this algorithm is also used in Theorem 5. In this data structure, each symbol is partitioned to various subpatterns determined by the index of the symbol along with predetermined random primes. Each subpattern is then fed to a dictionary matching algorithm. The dictionary entries are exactly the subpatterns of the original patterns and thus can be updated online.

The algorithm also needs to consider run-length encoding for each of these subpatterns in case they are highly periodic. It is clear that run-length encoding can be done independently for the pattern and the text.

Finally the approximation algorithm (Theorem 1.3 of [7]) uses a similar data structure to Theorem 5, but with different magnitudes for primes. Thus, the entire algorithm can be modified to run in an online fashion. ◀

3 Our Approach

Our approach to find all the k-periods of S is to first determine a set $\mathcal{T}$ of candidate k-periods, which is guaranteed to be a superset of all the true k-periods. We first describe the algorithm to find the k-period in two passes. In the first pass, we let $\mathcal{T}$ be the set of indices π that satisfy

$$\mathsf{HAM}\left(S[1, x], S[\pi + 1, \pi + x]\right) \le k,$$

for some appropriate value of x that we specify later. Note that by Observation 4, all k-periods must satisfy the above inequality. We show that even though $\mathcal{T}$ may be linear in size, we can succinctly represent $\mathcal{T}$ by adding a few additional indices into $\mathcal{T}$. We then show how to use the compressed version of $\mathcal{T}$ during the second pass to verify the candidates and output the true k-periods of S.

This strategy does not work if we are allowed only one pass; by the time we discover a candidate k-period p, it may be too late for us to start collecting the extra data needed to verify p (in the two-pass version this is not a problem, as the extra pass allows us to go back to the start of S and any needed data). We approach this problem by utilizing a trick from [12] of identifying candidate periods p using non-uniform criteria depending on the value of p. Using this idea, once a candidate period is found, it is not too late to verify that it is a true k-period, and the data can still be compressed into sublinear size.

Perhaps the biggest hidden challenge in the above approach is due to the major structural differences between exactly periodic and k-periodic strings; k-periodic strings show much less structure than exactly periodic strings. As a result, incremental adaptations of existing techniques on periodic strings do not yield corresponding schemes for k-periodic strings. In order to achieve small space, one needs to explore the weaker structural properties of k-periodic streams. A large part of the effort in this work is in formalizing said structure (see Appendix A), culminating in Theorem 23 and its proof, as well as exploring its application to our algorithms.

To show lower bounds for randomized algorithms finding the smallest k-period, we use a strategy similar to that in [12], using a reduction from the Augmented Index Problem. To show lower bounds for randomized algorithms finding the smallest k-period given the promise that the smallest k-period is at most $\frac{n}{2}$, we use Yao's Principle [28].

4 Two-Pass Algorithm to Compute k-Periods

In this section, we provide a two-pass, $\mathcal{O}\left(k^4 \log^9 n\right)$-space algorithm to output all k-periods of S. The general approach is to first identify a superset of the k-periods of S, based on the self-similarity of S, detected via the k-Mismatch algorithm of [7] as a black box. Unfortunately, while this tool allows us to match parts of S to each other, we get only incomplete information about possible periods, and this information is not readily stored in small space due to insufficient structure. We explore the structure of periods with mismatches in order to come up with a technique that massages our data into a form that can be compressed in small space, and is easily uncompressed. During the second pass, we go over S as well as the compressed data to verify the candidate periods.

We consider two classes of periods by their length, and run two separate algorithms in parallel. The first algorithm identifies all k-periods p with $p \leq \frac{n}{2}$, while the second algorithm identifies all k-periods p with $p > \frac{n}{2}$.

4.1 Finding small k-periods

Our algorithm for finding periods of length at most $n/2$ proceeds in two passes. In the first pass, we identify a set $\mathcal{T}$ of candidate k-periods, and formulate its compressed representation, $\mathcal{T}^C$. In the second pass, we recover each index from $\mathcal{T}^C$ and verify whether or not it is a k-period. We need $\mathcal{T}$ and $\mathcal{T}^C$ to satisfy four properties.

1. All true k-periods (likely accompanied by some candidate k-periods that are false positives) are in $\mathcal{T}$.
2. $\mathcal{T}^C$ can be stored in sublinear space.
3. $\mathcal{T}$ can be fully recovered from $\mathcal{T}^C$ in small space.
4. The verification process in the second pass weeds out those candidates that are not true periods in sublinear space.

We now describe our approach and show how it satisfies the above properties.

4.2 Pass 1: Property 1

We crucially observe that any k-period p must satisfy the requirement

$$\mathsf{HAM}\left(S[1,x], S[p+1,p+x]\right) \leq k$$

for all $x \leq n-p$, and specifically for $x = \frac{n}{2}$. This observation allows us to refer to indices as periods, as the index $p+1$ where the requirement is satisfied corresponds to (possible)

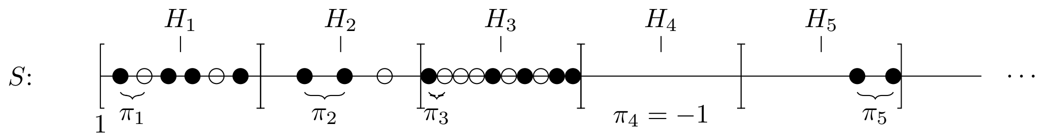

Figure 1 Observe that all dots in each interval are equally spaced after the first. These dots represent $\mathcal{T}^c$: the black dots represent $\mathcal{T}$, while the white dots are added to convert the irregularly spaced black dots into regularly spaced dot sequences.

k-period p. For the remainder of this algorithm, we set $x = \frac{n}{2}$, and designate the indices $p+1$ that satisfy the requirement with $x = \frac{n}{2}$ as candidate k-periods; collectively these indices serve as $\mathcal{T}$. Since satisfying this requirement is necessary but not sufficient for a candidate to be a real k-period, Property 1 follows.

4.3 Pass 1: Property 2

Observe that $\mathcal{T}$ could be linear in size, so we cannot store each index explicitly. We observe that if our indices followed an arithmetic progression, they could be kept implicitly in very succinct format (as is the case where there are no mismatches). Unfortunately, due to the presence of mismatches in S, such a regular structure does not happen. However, we show that it is still possible to implicitly add a small number of extra indices to our candidates and end up with an arithmetic series and allow for succinct representation. Our algorithm produces several such series, and represents each one in terms of its first index and the increment between consecutive terms, obtaining $\mathcal{T}^C$ from $\mathcal{T}$, with the details given below.

In order to compress $\mathcal{T}$ into $\mathcal{T}^C$, we partition $[1, x]$ into the $2mk + 2$ disjoint intervals $H_j = \left[\frac{jx}{2(mk+1)} + 1, \frac{(j+1)x}{2(mk+1)}\right)$, where $m = \log n$. The goal is, possibly through the addition of extra candidates, to represent the candidates in each interval as a single arithmetic series. This series will be represented by its first term, as well as the increment between its consecutive terms, π_j. As each new candidate arrives, we update π_j (except for the first update, π_j never increases, and it may shrink by an integer factor). Throughout the process, we maintain the invariant, by updating π_j, that the arithmetic sequence represented in H_j contains all candidates in H_j output by the k-Mismatch algorithm. Then it is clear that $\mathcal{T}^C$ and $\{\pi_j\}$ take sublinear space, satisfying Property 2.

4.4 Pass 1: Property 3

It remains to describe how to update π_j. The first time we see two candidates in H_j, we set π_j to be the increment between the candidates (before, it is set to -1). Each subsequent time we see a new candidate index in the interval H_j, we update π_j to be the greatest common divisor of π_j and the increment between the candidate and the smallest index in $\mathcal{T} \cap H_j$, which is kept explicitly. For instance, if our first candidate index is 10, and afterwards we receive 22, 26, 32 (assume the interval ends at 35), our π_j values over time are -1, 12, 4, 2. Ultimately, the candidates that we will be checking in Pass 2 will be 10, 12, 14, 16, 18, ..., 34. For another example, see Figure 1.

We now need to show that the above invariant is maintained throughout the algorithm. To do this, we show that any k-period $p \in H_j$ is an increment of some multiple of π_j away from the smallest index in $\mathcal{T} \cap H_j$. Then, if we insert implicitly into $\mathcal{T}$ *all indices* in H_j whose distance from the smallest index in $\mathcal{T} \cap H_j$ is a multiple of π_j, we will guarantee that any k-period in H_j will be included in $\mathcal{T}$.

We now show that any k-period p is implicitly represented in, and can be recovered from $\mathcal{T}^C$ and the values $\{\pi_j\}$ at the end of the first pass.

▶ **Lemma 8.** *If $p < \frac{n}{2}$ is a k-period and $p \in H_j$, then p can be recovered from $\mathcal{T}^C$ and π_j.*

Proof. Since $p \in H_j$ is a k-period, then it satisfies $\mathsf{HAM}\left(S[1, n-p], S[p+1, n]\right) \leq k$. More specifically, $i = p$ satisfies

$$\mathsf{HAM}\left(S\left[1, \frac{n}{2}\right], S\left[i+1, \frac{n}{2}+i\right]\right) \leq k$$

and will be reported by the k-Mismatch Algorithm. If there is no other index in $\mathcal{T}^C \cap H_j$, then p will be inserted into $\mathcal{T}^C$ in the first pass, so p can clearly be recovered from $\mathcal{T}^C$.

On the other hand, if there is another index q in $\mathcal{T}^C \cap H_j$, then π_j will be updated to be a divisor of the pairwise distances. Hence, the increment $p - q$ is a multiple of π_j. Any change that might later happen to π_j will be due to a gcd operation, and thus, will reduce it by a factor by at least 2. Thus, $p - q$ will remain a multiple of the final value of π_j, and p will be recovered at the end of the first pass as a member of $\mathcal{T}$. ◀

Thus Property 3 is satisfied. The first pass algorithm in full appears below.

(To determine any k-period p with $p \leq \frac{n}{2}$):

First pass:

1. Initialize $\pi_j = -1$ for each $0 \leq j < 2k \log n + 2$.
2. Initialize $\mathcal{T}^C = \emptyset$.
3. For each index i such that (using the k-Mismatch algorithm)

$$\mathsf{HAM}\left(S\left[1, \frac{n}{2}\right], S\left[i+1, \frac{n}{2}+i\right]\right) \leq k\,.$$

- For the integer j for which i is in the interval $H_j = \left[\frac{jn}{4(k \log n+1)} + 1, \frac{(j+1)n}{4(k \log n+1)}\right)$:
  a. If there exists no candidate $t \in \mathcal{T}^C$ in the interval H_j, then add i to $\mathcal{T}^C$.
  b. Otherwise, let t be the smallest candidate in $\mathcal{T}^C$ and either $\pi_j = -1$ or $\pi_j > 0$. If $\pi_j = -1$, then set $\pi_j = i - t$. Otherwise, set $\pi_j = \mathsf{gcd}\left(\pi_j, i - t\right)$.

4.5 Pass 2: Property 4

Our task in the second pass is to verify whether each candidate recovered from $\mathcal{T}^C$ and $\{\pi_j\}$ is actually a k-period or not. Thus, we must simultaneously check whether $\mathsf{HAM}\left(S[1, n-p], S[p+1, n]\right) \leq k$ for each candidate p, without using linear space. Fortunately, Theorem 9 states that at most $32k^2 \log n + 1$ unique fingerprints for substrings of length π_j are sufficient to recover the fingerprints of both $S[1, n-p]$ and $S[p+1, n]$ for any $p \in H_j$.

Before detailing, we first state a structural property, whose proof we defer to Appendix A. This property states that the greatest common divisor of the pairwise difference of any candidate k-periods within H_j must be a $(32k^2 \log n + 1)$-period.

▶ **Theorem 9.** *For some $0 \leq j < 2mk + 2$, let*

$$\mathcal{I}_j = \{i \in H_j \,|\, \mathsf{HAM}\left(S[1, x], S[i+1, i+x]\right) \leq k\}\,.$$

For any $p_1 < \ldots < p_m \in \mathcal{I}$, the greatest common divisor d of $p_2 - p_1, p_3 - p_1 \ldots, p_m - p_1$ satisfies

$$\mathsf{HAM}\,(S[1,x], S[d+1, d+x]) \leq 32mk^2 + 1.$$

Observe that π_j is exactly d. Moreover, each time the value of π_j changes, it gets divided by an integer factor at least equal to 2, ending up finally as a positive integer. Since $\pi_j \leq n$, this change can occur at most $\log n$ times, and so $m \leq \log n$. We now show that we can verify all candidates in sublinear space.

▶ **Lemma 10.** *Let p_i be a candidate k-period for a string S, with $p_1 < p_2 < \ldots < p_m$ all contained within H_j. Given the fingerprints of $S[1, n-p_1]$ and $S[p_1+1, n]$, we can determine whether or not S has k-period p_i for any $1 \leq i \leq m$ by storing at most $32k^2 \log n + 1$ additional fingerprints.*

Proof. Consider a decomposition of S into substrings w_i of length p_i, so that $S = w_1 \circ w_2 \circ w_3 \circ \ldots$. Note that each index i for which $w_i \neq w_{i+1}$ corresponds with at least one mismatch. It follows from Observation 2 that there exist at most k indices i for which $w_i \neq w_{i+1}$. Thus, recording the fingerprints and locations of these indices i suffice to determine whether or not there are k mismatches for candidate period p_i.

By Theorem 9, the greatest common divisor of the difference between each term in $\mathcal{I}$ is a $(32k^2 \log n + 1)$-period π_j. Thus, S can be decomposed $S = v \circ v_1 \circ v_2 \circ v_3 \circ \ldots$ so that v has length p_1, and each substring v_i has length π_j. It follows from Observation 2 that there exist at most $32k^2 \log n + 1$ indices i for which $v_i \neq v_{i+1}$. Therefore, recording the fingerprints and locations of these indices i allow us to recover the fingerprint of $S[1, n - p_i]$ from the fingerprint of $S[1, n - p_{i-1}]$, since $p_i - p_{i-1}$ is a multiple of π_j. Similarly, we can recover the fingerprint of $S[p_i + 1, n]$ from the fingerprint of $S[p_{i-1} + 1, n]$. Hence, we can confirm whether or not p_i is a k-period. ◀

The second pass algorithm in full follows.

(To determine all the k-periods p with $p \leq \frac{n}{2}$):

Second pass:

1. For each t such that $t \in \mathcal{T}^C$:
   - **a.** Let j be the integer for which t is in the interval $H_j = \left[\frac{jn}{4(k \log n+1)} + 1, \frac{(j+1)n}{4(k \log n+1)}\right)$
   - **b.** If $\pi_j > 0$, then record up to $32k^2 \log n + 1$ unique fingerprints of length π_j and of length t, starting from t.
   - **c.** Otherwise, record up to $32k^2 \log n + 1$ unique fingerprints of length t, starting from t.
   - **d.** Check if $\mathsf{HAM}\,(S[1, n-t], S[t+1, n]) \leq k$ and return t if this is true.
2. For each t which is in interval $H_j = \left[\frac{jn}{4(k \log n+1)} + 1, \frac{(j+1)n}{4(k \log n+1)}\right)$ for some integer j:
   - If there exists an index in $\mathcal{T}^C \cap H_j$ whose distance from t is a multiple of π_j, then check if $\mathsf{HAM}\,(S[1, n-t], S[t+1, n]) \leq k$ and return t if this is true.

This proves Property 4. Next, we show the correctness of the algorithm for small k-periods.

▶ **Lemma 11.** *For any k-period $p \leq \frac{n}{2}$, the algorithm outputs p.*

Proof. Since the intervals $\{H_j\}$ cover $\left[1, \frac{n}{2}\right]$, then $p \in H_j$ for some j. It follows from Lemma 8 that after the first pass, p can be recovered from $\mathcal{T}$ and π_j. Thus, the second pass tests whether or not p is a k-period. By Lemma 10, the algorithm outputs p, as desired. ◀

4.6 Finding large k-periods

As in the previous discussion, we would like to pick candidate periods during our first pass. However, if a k-period p satisfies $p > \frac{n}{2}$, then clearly it will no longer satisfy

$$\mathsf{HAM}\left(S\left[1, \frac{n}{2}\right], S\left[p+1, p+\frac{n}{2}\right]\right) \le k,$$

as $p + \frac{n}{2} > n$, and $S\left[p+\frac{n}{2}\right]$ is undefined. Instead, recall that $\mathsf{HAM}\left(S[1,x] = S[p+1, p+x]\right) \le k$ for all $x \le n-p$. Ideally, when choosing candidate periods p based on their satisfying this formula, we would like to use as large an x as possible without exceeding $n-p$, but we cannot do this without knowing the value of p. Instead, [12] observes we can try exponentially decreasing values of x: we run $\log n$ instances of the algorithm sequentially, with $x = \frac{n}{2}, \frac{n}{4}, \ldots$, since one of these values of x must be the largest one that does not lead to an illegal index of S. Therefore, the desired instance produces p, while all other instances do not.

(To determine a k-period p if $p > \frac{n}{2}$):

First pass:

1. Initialize $\pi_j^{(m)} = -1$ for each $0 \le j < 2k \log n + 2$ and $0 \le m \le \log n$.
2. Initialize $\mathcal{T}_m^C = \emptyset$.
3. For each index i, let r be the largest m such that $\frac{n}{2} + \frac{n}{4} + \ldots + \frac{n}{2^r} \le i$. Using the k-Mismatch algorithm, check whether

$$\mathsf{HAM}\left(S\left[1, \frac{n}{2^r}\right], S\left[i+1, i+\frac{n}{2^r}\right]\right) \le k.$$

If so, let $R = \frac{n}{2} + \frac{n}{4} + \ldots + \frac{n}{2^{r-1}}$ and j be the integer for which i is in the interval

$$H_j^{(r)} = \left[R + \frac{nj}{2^{r+1}(k \log n + 1)} + 1, R + \frac{n(j+1)}{2^{r+1}(k\log n + 1)}\right)$$

a. If there exists no candidate $t \in \mathcal{T}_r^C$ in the interval $H_j^{(r)}$, then add i to $\mathcal{T}_r^C$.

b. Otherwise, let t be the smallest candidate in $\mathcal{T}_r^C$ and either $\pi_j^{(r)} = -1$ or $\pi_j^{(r)} > 0$. If $\pi_j^{(r)} = -1$, then set $\pi_j^{(r)} = i - t$. Otherwise, set $\pi_j^{(r)} = \mathsf{gcd}\left(\pi_j^{(r)}, i-t\right)$.

This partition of $[1, n]$ into the disjoint intervals $\left[1, \frac{n}{2}\right]$, $\left[\frac{n}{2}+1, \frac{n}{2}+\frac{n}{4}\right]$, $\ldots$ guarantees that any k-period p is contained in one of these intervals. Moreover, the intervals $\{H_j^{(r)}\}$ partition

$$\left[\frac{n}{2} + \frac{n}{4} + \ldots + \frac{n}{2^{r-1}}, \frac{n}{2} + \ldots + \frac{n}{2^r}\right],$$

and so p can be recovered from $\mathcal{T}_r^C$ and $\{\pi_j^{(r)}\}$. We now present the algorithm for the second-pass to find all k-periods p for which $p > \frac{n}{2}$.

Second pass:

1. For each t and any r such that $t \in \mathcal{T}_r^C$:
   a. Let $R = \frac{n}{2} + \frac{n}{4} + \ldots + \frac{n}{2^{r-1}}$ and j be the integer for which t is in the interval
   $$H_j^{(r)} = \left[R + \frac{nj}{2^{r+1}(k\log n + 1)} + 1, R + \frac{n(j+1)}{2^{r+1}(k\log n+1)}\right)$$
   b. If $\pi_j^{(r)} > 0$, then record up to $32k^2\log n + 1$ unique fingerprints of length $\pi_j^{(r)}$ and of length t, starting from t.
   c. Otherwise, record up to $32k^2\log n + 1$ unique fingerprints of length t, starting from t.
   d. Check if HAM $(S[1, n-t], S[t+1, n]) \leq k$ and return t if this is true.
2. For each t which is in interval $H_j^{(r)} = \left[R + \frac{nj}{2^{r+1}(k\log n+1)} + 1, R + \frac{n(j+1)}{2^{r+1}(k\log n+1)}\right)$ for some integer j:
   a. If there exists an index in $\mathcal{T}_r^C \cap H_j^{(r)}$ whose distance from t is a multiple of $\pi_j^{(r)}$, then check if HAM $(S[1, n-t], S[t+1, n]) \leq k$ and return t if this is true.

Since correctness follows from the same arguments as the case where $p \leq \frac{n}{2}$, it remains to analyze the space complexity of our algorithm.

▶ **Theorem 12.** *There exists a two-pass algorithm that outputs all the k-periods of a given string using $\mathcal{O}\left(k^4\log^9 n\right)$ space.*

Proof. In the first pass, for each $\mathcal{T}_m$, we maintain a k-Mismatch algorithm which requires $\mathcal{O}\left(k^2\log^8 n\right)$ bits of space, as in Corollary 7. Since $1 \leq m \leq \log n$, we require $\mathcal{O}\left(k^2\log^9 n\right)$ bits of space in total. In the second pass, we keep up to $\mathcal{O}\left(k^2\log n\right)$ fingerprints for any set of indices in $\mathcal{T}_m$. Each fingerprint requires space $\mathcal{O}\left(k\log^6 n\right)$ and there may be $\mathcal{O}(k\log n)$ indices in $\mathcal{T}_m$ for each $1 \leq m \leq \log n$, for a total of $\mathcal{O}\left(k^4\log^7 n\right)$ bits of space. Thus, $\mathcal{O}\left(k^4\log^9 n\right)$ bits of space suffice for both passes. ◀

5 One-Pass Algorithm to Compute k-Periods

We now give a one-pass algorithm that outputs all the k-periods smaller than $\frac{n}{2}$. Similar to two-pass algorithm, we have two processes running in parallel. The first process handles all the k-periods p with $p \leq \frac{n}{4}$, while the second process handles the k-periods p with $p > \frac{n}{4}$. Both processes are designed again based on the crucial observation that all the k-periods p must satisfy HAM $(S[1, x], S[p+1, p+x]) \leq k$ for all $x \leq n-p$. In the first process, we set $x = \frac{n}{2}$ and find all indices i such that $S\left[i+1, i+\frac{n}{2}\right]$ has at most k mismatches from $S\left[1, \frac{n}{2}\right]$.

The second process cannot use the same approach, because the k-Mismatch Algorithm reports that index i is a candidate after reading position $\frac{n}{2} + i$, at which point we have already passed $n - i$. This means that the fingerprint of $S[1, n-i]$ cannot be built. For example, see Figure 2.

Thus, for a fixed p in the second process, if we set x to be the largest power of two which does not exceed $n - 2p$, the k-mismatch algorithm could report p. However, we cannot do this without knowing the value of p.

Building off the ideas in [12], we run $\log n$ instances of the algorithm in parallel, with $x = 1, 2, 4, \ldots$, then one of these values of x must correspond to the instance of k-mismatch algorithm that recognizes p and reports it for later verification.

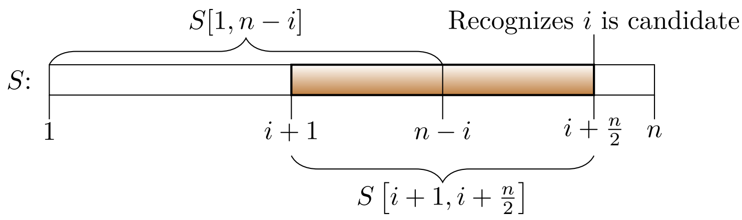

Figure 2 When i is recognized as a candidate, the algorithm has already passed $n-i$ and cannot build $S[1, n-i]$.

5.1 Finding small k-periods

We consider all the k-periods p with $p \leq \frac{n}{4}$ for this subsection. Run the k-Mismatch algorithm to find

$$\mathcal{T} = \left\{ i \,\middle|\, i \leq \frac{n}{4}, \mathsf{HAM}\left(S\left[1, \frac{n}{2}\right], S\left[i+1, i+\frac{n}{2}\right]\right) \leq k \right\}.$$

Upon finding an index $i \in \mathcal{T}$, the algorithm uses the fingerprint for $S\left[i+1, i+\frac{n}{2}\right]$ to continue building $S[i+1, n]$. Simultaneously, it builds $S[1, n-i]$, and checks whether $\mathsf{HAM}\left(S[1, n-i], S[i+1, n]\right) \leq k$. The algorithm identifies that $i \in \mathcal{T}$ upon reading character $i + \frac{n}{2} - 1$. Since $i \leq \frac{n}{4}$, then $i + \frac{n}{2} - 1 < \frac{3n}{4} \leq n - i$. Thus, the algorithm can identify i in time to build $S[1, n-i]$. By Theorem 9, these entries can be computed from a sequence of compressed fingerprints.

5.2 Finding large k-periods

Now, consider all the k-periods p with $\frac{n}{4} < p \leq \frac{n}{2}$. Let $I_m = \left[\frac{n}{2} - 2^m + 1, \frac{n}{2} - 2^{m-1}\right]$ and for $1 \leq m \leq \log n - 1$, define

$$\mathcal{T}_m = \{ i \,|\, i \in I_m, \mathsf{HAM}\left(S[1, 2^m], S[i+1, i+2^m]\right) \leq k \}.$$

Let π_m be a k-period of $S[1, 2^m]$. We first consider the case where $\pi_m \geq \frac{2^m}{4}$ and then the case where $\pi_m < \frac{2^m}{4}$.

▶ **Observation 13** ([7]). *If p is a k-period for $S[1, n/2]$, then each i such that*

$$\mathsf{HAM}\left(S\left[1, \frac{n}{2}\right], S\left[i+1, i+\frac{n}{2}\right]\right) \leq \frac{k}{2}$$

must be at least p symbols apart.

By Observation 13, if $\pi_m \geq \frac{2^m}{4}$, then $|\mathcal{T}_m| \leq 4$. Moreover, we can detect whether $i \in \mathcal{T}_m$ by index $\frac{n}{2} - 2^{m-1} + 2^m$. On the other hand, $n - i \geq \frac{n}{2} + 2^m + 1$, and so we can properly build $S[1, n-i]$.

Now, suppose $\pi_m < \frac{2^m}{4}$. Since $\mathcal{T}_m$ may be linear in size, we use the same trick to obtain a succinct representation, whose properties satisfy those in Section 4, while including a few additional indices. Let $S[2^m+1, 2^{m+1}] = w_1 w_2 \ldots w_t w'$, where each w_i has length π_m and for $0 \leq d \leq 3k$, let x_d be the largest index such that $S[1, 2^m] \circ w_1 \circ w_2 \circ \cdots \circ w_x$ has d-period π_m.

Let $\mathcal{T}_m = i_1, i_2, \ldots, i_r$ in increasing order. Let $S\left[i_r + 2^m + 1, \frac{n}{2} + 2^m\right] = v_1 v_2 \ldots v_s v'$, where each v_i has length π_m and let y be the largest index such that $S[i_r + 1, i_r + 2^m] \circ v_1 \circ v_2 \circ \cdots \circ v_y$ has $3k$-period π_m.

If $y = s$, then at most k of the substrings v_i can be unique by Observation 2. Moreover, by storing the fingerprints and positions of $\mathcal{O}\left(k^2 \log n\right)$ substrings, as well as v', we can recover the fingerprint of each $S[n-i_{j+1}, n-i_j]$ by Lemma 10. Thus, we keep the fingerprint of $S\left[\frac{n}{2}+1, n-i_r\right]$, and can construct the fingerprint of each $S\left[\frac{n}{2}+1, n-i_j\right]$

On the other hand if $y \neq s$, then for each i_j, let Δ be the number of indices z such that $i_j \leq z \leq i_r$ and $S[z] \neq S[z+\pi_m]$. That is, $\Delta = |\{z | i_j \leq z \leq i_r, S[z] \neq S[z+\pi_m]\}|$. Since π_m is a k-period of $S[1, 2^m]$, $\mathsf{HAM}\left(S[1, 2^m], S[i_j+1, i_j+2^m]\right) \leq k$, and each mismatch between $S[1, 2^m]$ and $S[i_j+1, i_j+2^m]$ can cause up to two indices z such that $S[z] \neq S[z+\pi_m]$, then it follows that $0 \leq \Delta \leq 3k$. Then if $y + |r-j| \neq x_{3k-\Delta}$, then $i_j \notin \mathcal{T}_m$, since $x_{3k-\Delta}$ is the largest index with $(3k-\Delta)$-period π_m, while y is the largest index with $3k$-period π_m.

Thus, for each $0 \leq \Delta \leq 2k$, there is at most one index j with $y + |r-j| \neq x_{2k+\Delta}$. Again by Lemma 10, we can compute the fingerprint of $S\left[\frac{n}{2}+1, n-i_j\right]$ by storing the fingerprints and positions of $\mathcal{O}\left(k^2 \log n\right)$ substrings.

Computing each x_d requires determining π_m and the fingerprint of $S[2^m - \pi_m + 1, 2^m]$. Since $\pi_m \leq \frac{2^m}{4}$, the algorithm determines π_m by position $\pi_m + 2^m < 2^m - \pi_m + 1$. Thus, the algorithm knows π_m in time to start creating the fingerprint of $S[2^m - \pi_m + 1, 2^m]$.

To compute y, we compute the fingerprint of $S[i_r+1, i_r+\pi_m]$. We then compute the fingerprint of each non-overlapping substring of length π_m starting from $i_r + \pi_m$, and compare the fingerprint to the previous fingerprint. We only record the fingerprint of the most recent substring, but keep a running count of the number of mismatches.

▶ **Theorem 14.** *There exists a one-pass algorithm that outputs all the k-periods p of a given string with $p \leq \frac{n}{2}$, and uses $\mathcal{O}\left(k^4 \log^9 n\right)$ bits of space.*

Proof. The process for small k-periods uses $\mathcal{O}\left(k^2 \log^8 n\right)$ bits of space determining $\mathcal{T}$. Verifying whether an index in $\mathcal{T}$ is actually a k-period requires the fingerprints of $\mathcal{O}\left(k^2 \log n\right)$ substrings, each using $\mathcal{O}\left(k \log^6 n\right)$ bits of space (Theorem 5). This adds up to a total of $\mathcal{O}\left(k^3 \log^7 n\right)$ bits of space.

The process for large k-periods has $\log n$ parallel instances of the k-Mismatch algorithm to compute $\mathcal{T}_m$ for $1 \leq m \leq \log n$, using $\mathcal{O}\left(k^2 \log^9 n\right)$ bits of space. To reconstruct the fingerprint of $S[1, n-i]$ for each $i \in \mathcal{T}_m$ the algorithm needs to store the fingerprints of at most $\mathcal{O}\left(k^2 \log n\right)$ unique substrings (Lemma 10). Each fingerprint uses $\mathcal{O}\left(k \log^6 n\right)$ bits of space (Theorem 5) and there can be up to $\mathcal{O}\left(k \log n\right)$ indices in $\mathcal{T}_m$. This adds up to a total of $\mathcal{O}\left(k^4 \log^9 n\right)$ bits of space.

Thus, $\mathcal{O}\left(k^4 \log^9 n\right)$ bits of space suffice for both processes. ◀

6 Lower Bounds

6.1 Lower Bounds for General Periods

Recall the following variant of the Augmented Indexing Problem, denoted $\mathsf{IND}_{n,\delta}$, where Alice is given a string $S \in \Sigma^n$. Bob is given an index $i \in [n]$, as well as $S[1, i-1]$, and must output $S[i]$ correctly with probability at least $1-\delta$.

▶ **Lemma 15** ([24]). *The one-way communication complexity of* $\mathsf{IND}_{n,\delta}$ *is* $\Omega((1-\delta)n \log |\Sigma|)$.

▶ **Theorem 16.** *Any one-pass streaming algorithm which computes the smallest k-period of an input string S requires $\Omega(n)$ space.*

Proof. Consider the following communication game between Alice and Bob, who are given strings A and B respectively. Both A and B have length n, and the goal is to compute the smallest k-period of $a \circ b$. Then we show that any one-way protocol which successfully computes the smallest k-period of $a \circ b$ requires $\Omega(n)$ communication by a reduction from the augmented indexing problem.

Suppose Alice gets a string $S \in \{0,1\}^n$, while Bob gets an index $i \in [n-1]$ and $S[1, i-1]$. Let $\mathbf{u}$ be the binary negation of $S[1]$, i.e., $\mathbf{u} = 1 - S[1]$. Then Alice sets $A = (S[1])^k(S[2])^k \ldots (S[n])^k$ and Bob sets $B = \mathbf{u}^{k(n-i)} \circ (S[1])^k(S[2])^k \ldots (S[i-1])^k \circ \mathbf{1}^k$ so that both A and B have length kn. Moreover, the smallest k-period of $A \circ B$ is $k(2n-i)$ if and only if $S[i] = 1$. ◀

6.2 Lower Bounds for Small Periods

We now show that for $k = o(\sqrt{n})$, even given the promise that the smallest k-period is at most $\frac{n}{2}$, any randomized algorithm which computes the smallest k-period with probability at least $1 - \frac{1}{n}$ requires $\Omega(k \log n)$ space. By Yao's Minimax Principle [28], it suffices to show a distribution over inputs such that every deterministic algorithm using less than $\frac{k \log n}{6}$ bits of memory fails with probability at least $\frac{1}{n}$.

Define an infinite string $1^1 0^1 1^2 0^2 1^3 0^3 \ldots$, as in [16], and let ν be the prefix of length $\frac{n}{4}$. Let X be the set of binary strings of length $\frac{n}{4}$ at Hamming distance $\frac{k}{2}$ from ν. Given $x \in X$, let Y_x be the set of binary strings of length $\frac{n}{4}$ with either $\mathsf{HAM}\,(x,y) = \frac{k}{2}$ or $\mathsf{HAM}\,(x,y) = \frac{k}{2}+1$. We pick (x,y) uniformly at random from (X, Y_x).

▶ **Theorem 17.** *Given an input $x \circ y$, any deterministic algorithm $\mathcal{D}$ that uses less than $\frac{k \log n}{6}$ bits of memory cannot correctly output whether $\mathsf{HAM}\,(x,y) = \frac{k}{2}$ or $\mathsf{HAM}\,(x,y) > \frac{k}{2}$ with probability at least $1 - \frac{1}{n}$, for $k = o(\sqrt{n})$.*

Proof. Note that $|X| = \binom{n/4}{k/2}$. By Stirling's approximation, $|X| \geq \left(\frac{n}{2k}\right)^{k/2} \geq \left(\frac{n}{4}\right)^{k/4}$ for $k = o(\sqrt{n})$.

Because $\mathcal{D}$ uses less than $\frac{k \log n}{6}$ bits of memory, then $\mathcal{D}$ has at most $2^{\frac{k \log n}{6}} = n^{k/6}$ unique memory configurations. Since $|X| \geq \left(\frac{n}{4}\right)^{k/4}$, then there are at least $\frac{1}{2}(|X| - n^{k/6}) \geq \frac{|X|}{4}$ pairs x, x' such that $\mathcal{D}$ has the same configuration after reading x and x'. We show that $\mathcal{D}$ errs on a significant fraction of these pairs x, x'.

Let $\mathcal{I}$ be the positions where either x or x' differ from ν, so that $\frac{k}{2}+1 \leq |\mathcal{I}| \leq k$. Observe that if $\mathsf{HAM}\,(x,y) = \frac{k}{2}$, but x and y do not differ in any positions of $\mathcal{I}$, then $\mathsf{HAM}\,(x',y) > \frac{k}{2}$. Recall that $\mathcal{D}$ has the same configuration after reading x and x', so then $\mathcal{D}$ has the same configuration after reading $x \circ y$ and $x' \circ y$. But since $\mathsf{HAM}\,(x,y) = \frac{k}{2}$ and $\mathsf{HAM}\,(x',y) > \frac{k}{2}$, then the output of $\mathcal{D}$ is incorrect for either $x \circ y$ or $x' \circ y$.

For each pair (x, x'), there are $\binom{n/4-|\mathcal{I}|}{k/2} \geq \binom{n/4-k}{k/2}$ such y with $\mathsf{HAM}\,(x,y) = \frac{k}{2}$, but x and y do not differ in any positions of $\mathcal{I}$. Hence, there are $\frac{|X|}{4}\binom{n/4-k}{k/2}$ strings $S(x,y)$ for which $\mathcal{D}$ errs. Recall that y satisfies either $\mathsf{HAM}\,(x,y) = \frac{k}{2}$ or $\mathsf{HAM}\,(x,y) = \frac{k}{2}+1$ so that there are $|X|\left(\binom{n/4}{k/2} + \binom{n/4}{k/2+1}\right)$ strings $x \circ y$ in total. Thus, the probability of error is at least

$$\begin{aligned}
\frac{\frac{|X|}{4}\binom{n/4-k}{k/2}}{|X|\left(\binom{n/4}{k/2} + \binom{n/4}{k/2+1}\right)} &= \frac{1}{4} \cdot \frac{\binom{n/4-k}{k/2}}{\binom{n/4+1}{k/2+1}} = \frac{(k/2+1)}{4}\frac{(n/4-3k/2+1)\ldots(n/4-k)}{(n/4-k/2+1)\ldots(n/4+1)} \\
&\geq \frac{k/2+1}{n+4}\left(\frac{n/4-3k/2+1}{n/4-k/2+1}\right)^{k/2} = \frac{k+2}{2n+8}\left(1 - \frac{k}{n/4-k/2+1}\right)^{k/2} \\
&\geq \frac{k+2}{2n+8}\left(1 - \frac{k^2}{n/2-k+2}\right) \geq \frac{1}{n}
\end{aligned}$$

where the last line holds for large n, from Bernoulli's Inequality and $k = o(\sqrt{n})$. ◀

▶ **Lemma 18.** *For $k = o(\sqrt{n})$, any k-period of the string $S(x, y) = x \circ y \circ x \circ x$ is at least $\frac{n}{4}$.*

Proof. We show that stronger result that if $p < \frac{n}{4}$, $k > 2$, and $n > 4(18k+1)(18k+2)$, then $|\{z|S[z] \neq S[z+p]\}| > \sqrt{\frac{n}{8}} > k$, for $k = o(\sqrt{n})$.

Let $T = \nu \circ \nu \circ x \circ x$ and for each z, consider $T[z]$ and $T[z+p]$. For each $j > 0$, some position $z+p$ in $1^{2j}0^{2j}1^{2j+1}0^{2j+1}$ in the second ν corresponds with a mismatch in z. Since $\mathsf{HAM}(x, \nu) = \frac{k}{2}$ and $\mathsf{HAM}(x, y) \leq \frac{k}{2} + 1$, then $\mathsf{HAM}\left(S\left[1, \frac{n}{2}\right], T\left[1, \frac{n}{2}\right]\right) \leq \frac{3k}{2} + 1$. Each mismatch between S and T can cause at most two indices z for which $T[z] \neq T[z+p]$ but $S[z] = S[z+p]$. Thus, by setting $j = 6k > 2\left(\frac{3k}{2} + 1\right) + 2k$, we have that for $\frac{n}{4} > (12k+1)(12k+2)$, there are at least $6k$ indices z for which $T[z] \neq T[z+p]$, and thus at least $2k$ indices for which $S[z] \neq S[z+p]$. ◀

▶ **Corollary 19.** *If $\mathsf{HAM}(x, y) = \frac{k}{2}$, then the string $S(x, y) = x \circ y \circ x \circ x$ has period $\frac{n}{4}$. On the other hand, if $\mathsf{HAM}(x, y) = \frac{k}{2} + 1$, then $S(x, y)$ has period greater than $\frac{n}{4}$.*

▶ **Theorem 20.** *For $k = o(\sqrt{n})$ with $k > 2$, any one-pass streaming algorithm which computes the smallest k-period of an input string S with probability at least $1 - \frac{1}{n}$ requires $\Omega(k \log n)$ space, even under the promise that the k-period is at most $\frac{n}{2}$.*

Proof. By Theorem 17, any algorithm using less than $\frac{k \log n}{6}$ bits of memory cannot distinguish between $\mathsf{HAM}(x, y) = \frac{k}{2}$ and $\mathsf{HAM}(x, y) = \frac{k}{2} + 1$ with probability at least $1 - 1/n$. Thus, no algorithm can distinguish whether the period of $S(x, y)$ is $\frac{n}{4}$ with probability at least $1 - 1/n$ while using less than $\frac{k \log n}{6}$ bits of memory. ◀

References

1 Amihood Amir, Estrella Eisenberg, and Avivit Levy. Approximate periodicity. *Algorithms and Computation*, pages 25–36, 2010.

2 Alexandr Andoni, Assaf Goldberger, Andrew McGregor, and Ely Porat. Homomorphic fingerprints under misalignments: sketching edit and shift distances. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 931–940, 2013.

3 Alberto Apostolico and Zvi Galil, editors. *Pattern Matching Algorithms*. Oxford University Press, Oxford, UK, 1997.

4 Petra Berenbrink, Funda Ergün, Frederik Mallmann-Trenn, and Erfan Sadeqi Azer. Palindrome recognition in the streaming model. In *31st International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 149–161, 2014.

5 Raphaël Clifford, Klim Efremenko, Ely Porat, and Amir Rothschild. From coding theory to efficient pattern matching. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 778–784, 2009.

6 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana A. Starikovskaya. Dictionary matching in a stream. In *Algorithms – ESA 23rd Annual European Symposium, Proceedings*, pages 361–372, 2015.

7 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana A. Starikovskaya. The k-mismatch problem revisited. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2039–2052, 2016.

8 Raphaël Clifford, Markus Jalsenius, Ely Porat, and Benjamin Sach. Space lower bounds for online pattern matching. *Theoretical Computer Science*, 483:68–74, 2013.

9 Michael S. Crouch and Andrew McGregor. Periodicity and cyclic shifts via linear sketches. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques – 14th International Workshop, APPROX, and 15th International Workshop, RANDOM. Proceedings*, pages 158–170, 2011.

10 Mohamed G. Elfeky, Walid G. Aref, and Ahmed K. Elmagarmid. STAGGER: periodicity mining of data streams using expanding sliding windows. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM)*, pages 188–199, 2006.

11 Funda Ergün, Elena Grigorescu, Erfan Sadeqi Azer, and Samson Zhou. Streaming periodicity with mismatches, 2017. URL: http://homes.soic.indiana.edu/fergun/PUBLICATIONS/mismatchperiodicity.pdf.

12 Funda Ergün, Hossein Jowhari, and Mert Saglam. Periodicity in streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 13th International Workshop, APPROX 2010, and 14th International Workshop, RANDOM 2010. Proceedings*, pages 545–559, 2010.

13 Funda Ergün, S. Muthukrishnan, and Süleyman Cenk Sahinalp. Periodicity testing with sublinear samples and space. *ACM Trans. Algorithms*, 6(2):43:1–43:14, 2010.

14 Zvi Galil and Joel Seiferas. Time-space-optimal string matching. *Journal of Computer and System Sciences*, 26(3):280–294, 1983.

15 Pawel Gawrychowski. Optimal pattern matching in LZW compressed strings. *ACM Transactions on Algorithms (TALG)*, 9(3):25, 2013.

16 Pawel Gawrychowski, Oleg Merkurev, Arseny M. Shur, and Przemyslaw Uznanski. Tight tradeoffs for real-time approximation of longest palindromes in streams. In *27th Annual Symposium on Combinatorial Pattern Matching, CPM*, pages 18:1–18:13, 2016.

17 Shay Golan, Tsvi Kopelowitz, and Ely Porat. Streaming Pattern Matching with d Wildcards. In *24th Annual European Symposium on Algorithms (ESA)*, pages 44:1–44:16, 2016.

18 Elena Grigorescu, Erfan Sadeqi Azer, and Samson Zhou. Streaming for aibohphobes: Longest palindrome with mismatches. *CoRR*, abs/1705.01887, 2017. URL: http://arxiv.org/abs/1705.01887.

19 Piotr Indyk, Nick Koudas, and S. Muthukrishnan. Identifying representative trends in massive time series data sets using sketches. In *VLDB, Proceedings of 26th International Conference on Very Large Data Bases*, pages 363–372, 2000.

20 Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.

21 Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977.

22 Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM journal on computing*, 6(2):323–350, 1977.

23 Oded Lachish and Ilan Newman. Testing periodicity. *Algorithmica*, 60(2):401–420, 2011.

24 Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, pages 103–111, 1995.

25 Benny Porat and Ely Porat. Exact and approximate pattern matching in the streaming model. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 315–323, 2009.

26 Ely Porat and Ohad Lipsky. Improved sketching of hamming distance with error correcting. In *Annual Symposium on Combinatorial Pattern Matching*, pages 173–182, 2007.

27 Jakub Radoszewski and Tatiana Starikovskaya. Streaming k-mismatch with data recovery and applications. *arXiv preprint arXiv:1607.05626*, 2016.

28 Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *18th Annual Symposium on Foundations of Computer Science, FOCS*, pages 222–227, 1977.

A Structural Properties of k-Periodic Strings

In this section, we show several steps towards proving Theorem 9. We defer the detailed proofs to the full version [11].

We first show Theorem 23, which assumes there are only two candidate k-periods and both are small. We then relax these conditions and prove Theorem 30, which does not restrict the number of candidate k-periods, but still assumes that their magnitudes are small. Theorem 9 considers all candidate k-periods in some interval. We use the fact that the *difference* between these candidates is small, thus meeting the conditions of Theorem 30, although with an increase in the number of mismatches.

To show that the greatest common divisor d of any two reasonably small candidates $p < q$ for k-periods is also a $(16k^2 + 1)$-period (Theorem 23), we consider the cases where either all candidates are less than $(2k + 1)d$ (Lemma 24) or some candidate is at least $(2k + 1)d$ (Lemma 25).

In the first case, where all candidate period are less than $(2k + 1)d$, we partition the string into disjoint intervals of a certain length, followed by partitioning the intervals further into congruence classes. We show in Lemma 22 that any partition which contains an index i such that $S[i] \neq S[i + d]$ must also contain an index j which is a mismatch from some symbol p or q distance away. Since there are at most $2k$ indices j, we can then bound the number of such partitions, and then extract an upper bound on the number of such indices i.

In the second case, where some candidate is at least $(2k + 1)d$, our argument relies on forming a grid (such as in Figure 3) where adjacent points are indices which either differ by p or q. We include $2k + 1$ rows and columns in this grid. Since $\frac{q}{d} \geq 2k + 1$, then no index in S is represented by multiple points in the grid. We call an edge between adjacent points "bad" if the two corresponding indices form a mismatch.

▶ **Observation 21.** *$S[i] \neq S[i + d]$ only if each path between i and $i + d$ contains a bad edge.*

Our grid contains at most $2k$ bad edges, since p and q are both k-periods, and each index is represented at most once. We then show that for all but at most $(16k^2 + 1)$ indices i, there exists a path between indices i and $i + d$ that avoids bad edges. Therefore, there are at most $(16k^2 + 1)$ indices i such that $S[i] \neq S[i + d]$, which shows that d is an $(16k^2 + 1)$-period.

Before proving Lemma 24, we first show a number theoretic result that given integers i, p, q, we can repeatedly hop by distance p or q, starting from i, ending at $i + \mathsf{gcd}(p, q)$, all the while staying in a "small" interval.

▶ **Lemma 22.** *Suppose $p < q$ are two positive integers with $\mathsf{gcd}(p, q) = d$. Let i be an integer such that $1 \leq i \leq p + q - d$. Then there exists a sequence of integers $i = t_0, \ldots, t_m = i + d$ where $|t_i - t_{i+1}|$ is either p or q, and $1 \leq t_i < p + q$. Furthermore, each integer is congruent to $i \pmod d$. In other words, any interval of length $p + q$ which contains indices $i, i + d$ such that $S[i] \neq S[i + d]$ also contains an index j such that either $S[j] \neq S[j + p]$ or $S[j] \neq S[j + q]$.*

Proof. Since d is the greatest common divisor of p and q, then there exist integers a, b such that $ap + bq = d$. Suppose $a > 0$. Then consider the sequence $t_i = t_{i-1} + p$ if $1 \leq t_{i-1} \leq q$. Otherwise, if $t_{i-1} > q$, let $t_i = t_{i-1} - q$. Then clearly, each $|t_i - t_{i+1}|$ is either p or q, and $1 \leq t_i < p + q$. That is, each t_i either increases the coefficient of p by one, or decreases the coefficient of q by one. Thus, at the last time the coefficient of p is a, $t_i = ap + bq = d$, since any other coefficient of q would cause either $t_i > q$ or $t_i < 1$. Hence, terminating the sequence at this step produces the desired output, and a similar argument follows if $b > 0$ instead of $a > 0$. Since $p \equiv q \equiv 0 \pmod d$, then all integers in these sequence are congruent to $i \pmod d$. ◀

We now prove that the greatest common divisor d of any two reasonably small candidates p, q for k-periods is also a $(16k^2+1)$-period.

▶ **Theorem 23.** *For any $1 \le x \le \frac{n}{2}$, let $\mathcal{I} = \left\{ i \,\middle|\, i \le \frac{x}{4k+2}, \mathsf{HAM}\left(S[1,x], S[i+1, i+x]\right) \le k \right\}$. For any two $p, q \in \mathcal{I}$ with $p < q$, their greatest common divisor, $d = \mathsf{gcd}\left(p, q\right)$ satisfies*

$$\mathsf{HAM}\left(S[1,x], S[d+1, d+x]\right) \le (16k^2+1).$$

We now proceed to the proof of Theorem 23 for the case $q < (2k+1)d$.

▶ **Lemma 24.** *Theorem 23 holds when $q < (2k+1)d$.*

Proof. If $x \le 16k^2$, then clearly there are at most $16k^2$ indices i such that $S[i] \ne S[i+d]$, and so d is a $(16k^2+1)$-period. Otherwise, suppose $x > 16k^2+1$, and by way of contradiction, that there are at least $16k^2+1$ indices i such that $S[i] \ne S[i+d]$.

Consider the following two classes of intervals of length $\frac{p+q}{2}$:

$$\mathcal{I}_1 = \left[1, \frac{p+q}{2}\right], \left[p+q+1, \frac{3(p+q)}{2}\right], \left[2(p+q)+1, \frac{5(p+q)}{2}\right], \ldots$$

and

$$\mathcal{I}_2 = \left[\frac{p+q}{2}+1, p+q\right], \left[\frac{3(p+q)}{2}+1, 2(p+q)\right], \left[\frac{5(p+q)}{2}+1, 3(p+q)\right], \ldots.$$

If there are at least $16k^2+1$ indices i such that $S[i] \ne S[i+d]$, then either $\mathcal{I}_1$ or $\mathcal{I}_2$ contains at least $8k^2+1$ of these indices.

Suppose $\mathcal{I}_1$ has at least $8k^2+1$ indices i such that $S[i] \ne S[i+d]$. Now, consider the disjoint intervals of length $p+q$: $[1, p+q]$, $[p+q+1, 2(p+q)]$, $[2(p+q)+1, 3(p+q)]$, ... Furthermore, for each of these intervals, consider the congruence classes modulo d. Since $x > 16k^2+1$ and each of these congruence classes within an intervals have $\frac{p+q}{d} < \frac{2q}{d} \le 2(2k) = 4k$ indices, then $S[1,x]$ certainly contains at least $2k+1$ of these congruence classes.

If $\mathcal{I}_1$ has at least $8k^2+1$ indices i such that $S[i] \ne S[i+d]$ and each congruence class within an interval contains less than $4k$ indices, then there are at least $2k+1$ congruence classes containing such an index i. Because each of these indices occur within $\mathcal{I}_1$, it follows that both i and $i+d$ are contained within the interval (and therefore, the same congruence class). By Lemma 22, each congruence class within an interval containing indices i and $i+d$ $S[i] \ne S[i+d]$ also contains an index j such that either $S[j] \ne S[j+p]$ or $S[j] \ne S[j+q]$. Since there are at least $2k+1$ congruence classes within intervals, then there are at least $2k+1$ such indices j. This either contradicts that there are at most k indices j such that $S[j] \ne S[j+p]$ or there are at most k indices j such that $S[j] \ne S[j+q]$.

The proof for the case where $\mathcal{I}_2$ has at least $8k^2+1$ indices i such that $S[i] \ne S[i+d]$ is symmetric. ◀

The following lemma considers the case where at least one of candidate periods p or q is at least $(2k+1)d$. Without loss of generality, assume $q \ge (2k+1)d$. We form a grid, such as in Figure 3, where adjacent points in the grid correspond to indices which either differ by p or q. An edge between adjacent points is "bad" if the two corresponding indices form a mismatch.

From Observation 21, $S[i] \ne S[i+d]$ only if each path between i and $i+d$ contains a bad edge. Thus, if $S[i] \ne S[i+d]$, then the point in the grid corresponding to i must be contained in some region whose boundary is formed by bad edges. We partition the indices into congruence classes modulo d, count the number of mismatches in each class, and aggregate the results.

That is, in a particular congruence class, we assume p is a k_1-period, and q is a k_2-period, where $k_1, k_2 \leq k$. Then the grid contains at most $k_1 + k_2$ bad edges, which bounds the perimeter of the regions. From this, we deduce a generous bound of $(16k_1k_2 + 1)$ on the number of points inside these regions, which is equivalent to the number of indices i such that $S[i] \neq S[i+d]$ in the congruence class. We then aggregate over all congruence classes to show that d is a $(16k^2 + 1)$-period.

▶ **Lemma 25.** *Let $p \leq q$ and k be positive integers with $q \geq (2k+1)d$ and let $d = \mathsf{gcd}\,(p, q)$. Given a string S and an integer $0 \leq m < d$, let there be $k_1 > 0$ indices $i \equiv m \pmod{d}$ such that $S[i] \neq S[i+p]$ and $k_2 > 0$ indices $i \equiv m \pmod{d}$, not necessarily disjoint, such that $S[i] \neq S[i+q]$ and $k_1, k_2 \leq k$. If $d = \mathsf{gcd}\,(p, q)$, then there exist at most $8k_1k_2 + 1$ indices $i \equiv m \pmod{d}$ such that $S[i] \neq S[i+d]$.*

Proof. Consider a pair of indices $(i, i+d)$ with $S[i] \neq S[i+d]$ in congruence class $m \pmod{d}$. We ultimately want to build a grid of "large" size around i, but this may result in illegal indices if i is too small or too large. Therefore, we first consider the case where $k(p+q) \leq i \leq x - k(p+q)$, where we can place i in the center of the grid. We then describe a similar argument with modifications for $i < k(p+q)$ or $i > x - k(p+q)$, when we must place i near the periphery of the grid.

Given index i with $k(p+q) \leq i \leq x - k(p+q)$, we define a grid on a subset of indices of $S[1, x]$. The node at the center is i and for any node j, the nodes $j+p$, $j+q$, $j-p$ and $j-q$ are the top, right, bottom and left neighbors of j, respectively. See Figure 3 for example of such a grid.

We include $(2k+1)$ rows and columns in this grid, where i is the intersection of the middle row and the middle column. Note that since $k(p+q) \leq i \leq x - k(p+q)$, all points in the grid correspond to indices of S.

▶ **Claim 26.** *No indices of S correspond to multiple points in the grid.*

Proof. Suppose, by way of contradiction, there exists some index j which is represented by multiple points in the grid. That is, $j = i + a_1p + b_1q = i + a_2p + b_2q$ with $a_1 \neq a_2$. Since $d = \mathsf{gcd}\,(p, q)$, there exist integers r, s with $p = rd$, $q = sd$, and $\mathsf{gcd}\,(r, s) = 1$. Then $(a_1 - a_2)p = (b_2 - b_1)q$ so $(a_1 - a_2)r = (b_2 - b_1)s$. Because $\mathsf{gcd}\,(r, s) = 1$, it follows that $(a_1 - a_2)$ is divisible by $s = \frac{q}{d} \geq 2k+1$. Therefore, $|a_1 - a_2| \geq 2k+1$, and so a_1 and a_2 are at least $2k+1$ columns apart. However, this contradicts both points being in the grid, since the grid contains exactly $2k+1$ columns. ◀

▶ **Claim 27.** *There exist at least $k+1$ rows and $k+1$ columns in the grid that do not contain any bad edge.*

Proof. Since $\mathsf{HAM}\,(S[1, x], S[\alpha+1, \alpha+x]) \leq k$, for $\alpha = p, q$, there are at most k indices i for which $S[i] \neq S[i+p]$ or $S[i] \neq S[i+q]$. By Claim 26, each index is represented at most once. Hence, there are at most k vertical bad edges and at most k horizontal bad edges in this grid. Because the grid contains $2k+1$ rows and columns, then there exist at least $k+1$ rows and columns in the grid that do not contain any bad edge. ◀

We call these rows and columns *no-change*.

▶ **Claim 28.** *If there exists a path between i and a no-change row or column in a grid containing i avoiding bad edges, and a path between $i+d$ and a no-change row or column in a grid containing $i+d$ avoiding bad edges, then there exists a path between i and $i+d$ avoiding bad edges.*

Proof. Notice that some no-change row in the grid centered at i must also be a no-change row in the grid centered at $i+q$, since there are at least $k+1$ no-change rows in each grid, but the two grids overlap in $2k+1$ rows. Similarly, some no-change column in the grid centered at i must also be a no-change row in the grid centered at $i+p$. These common no-change rows and columns allow traversal between grids, as we can freely traverse between any no-change rows and columns while avoiding bad edges. Thus, if we can traverse from i to any no-change row in the first grid, we can ultimately reach any no-change row in the final grid containing $i+d$ while avoiding all bad edges. Finally, if we can traverse between $i+d$ and any no-change row in the final grid, then there exists a path between i and $i+d$ without any bad edges. ◀

This construction describes a possible path from i to $i+d$ with the help of these no-change rows and columns between grids. Notice that it is possible that there is no path from i to $i+d$ simply because a lot of bad edges have surrounded node i or $i+d$. (This is a necessary but not sufficient condition.)

We use the term *isolated* node, to describe any node which is in a region enclosed by bad edges. Note that points in such enclosed regions are also possibly part of mismatched indices $(j, j+d)$. We argue that the most number of unique indices which can enclosed with k_1 vertical edges and k_2 horizontal edges is $\frac{k_1k_2}{2} + 2k_1 + 2k_2$, even on an extended grid with no boundaries and multiple vertices/edges which correspond to the same index.

▶ **Claim 29.** *The number of isolated nodes is at most $\frac{k_1k_2}{2} + 2k_1 + 2k_2$.*

We sketch the details of the proof of Claim 29, with full details provided in [11]. The total area of regions enclosed by at most k_1 vertical bad edges and at most k_2 horizontal bad edges is at most $\frac{k_1k_2}{4}$. Thus, the number of isolated nodes cannot exceed $\frac{k_1k_2}{4}$.

The number of $(i, i+d)$ mismatches is at most double the number of isolated nodes (if i is isolated, both $(i, i+d)$ and $(i-d, i)$ may be mismatches) plus the number of mismatched edges. The former is bounded by $\frac{k_1k_2}{4}$, the latter by k_1+k_2. See Figure 3 for example.

We defer the casework for $i < k(p+q)$ and $i > x - k(p+q)$ to the full version [11]. ◀

The proof of Theorem 23 follows by aggregating each congruence class with mismatched indices, handled in Lemma 25.

We generalize Theorem 23 by showing that the greatest common divisor of any $m \geq 2$ reasonably small candidates for k-periods is also a $(2mk^2+1)$-period. We emphasize that it is sufficient for $m \leq \log n$, since the greatest common divisor can change at most $\log n$ times.

▶ **Theorem 30.** *Let $\mathcal{I} = \left\{ i \,\middle|\, i \leq \frac{x}{2(mk+1)}, \mathsf{HAM}\left(S[1,x], S[i+1, i+x]\right) \leq k \right\}$. The greatest common divisor of any $p_1, \ldots, p_m \in \mathcal{I}$, $d = \mathsf{gcd}\left(p_1, \ldots, p_m\right)$, satisfies*

$$\mathsf{HAM}\left(S[1,x], S[d+1, d+x]\right) \leq 8mk^2 + 1.$$

Although the pairwise greatest common divisor between two candidates p_i and p_j is no longer d, considering $\delta = \mathsf{gcd}\left(p_1, p_m\right)$ suffices for the analysis. If $\frac{p_m}{\delta} < 2k+1$, then the proof is similar to that of Lemma 24. Otherwise if $\frac{p_m}{\delta} \geq 2k+1$, the proof is similar to that of Lemma 25. We show a k^2 bound on the volume of an enclosed region, whose surface area

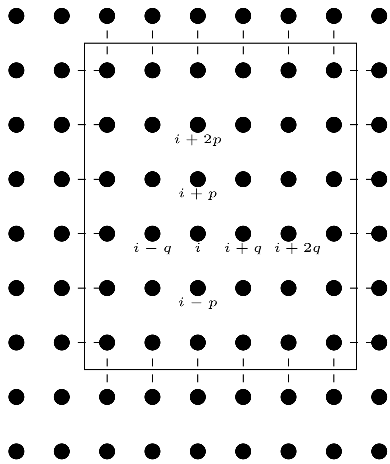

Figure 3 The dashed lines are bad edges. The total area of the enclosed regions can be at most k^2 if the perimeter is at most $4k$.

is at most mk, within a hypergrid. This yields a related bound on the number of isolated nodes.

Observe that Theorem 9 relaxes the constraints of Theorem 30. The full details for the proof of Theorem 23, Theorem 30, and Theorem 9 are provided in [11].